Computer Networking

COMP 177 | Fall 2020 | University of the Pacific | Jeff Shafer

HTTP

Hypertext Transport Protocol

Recap

Past Topics

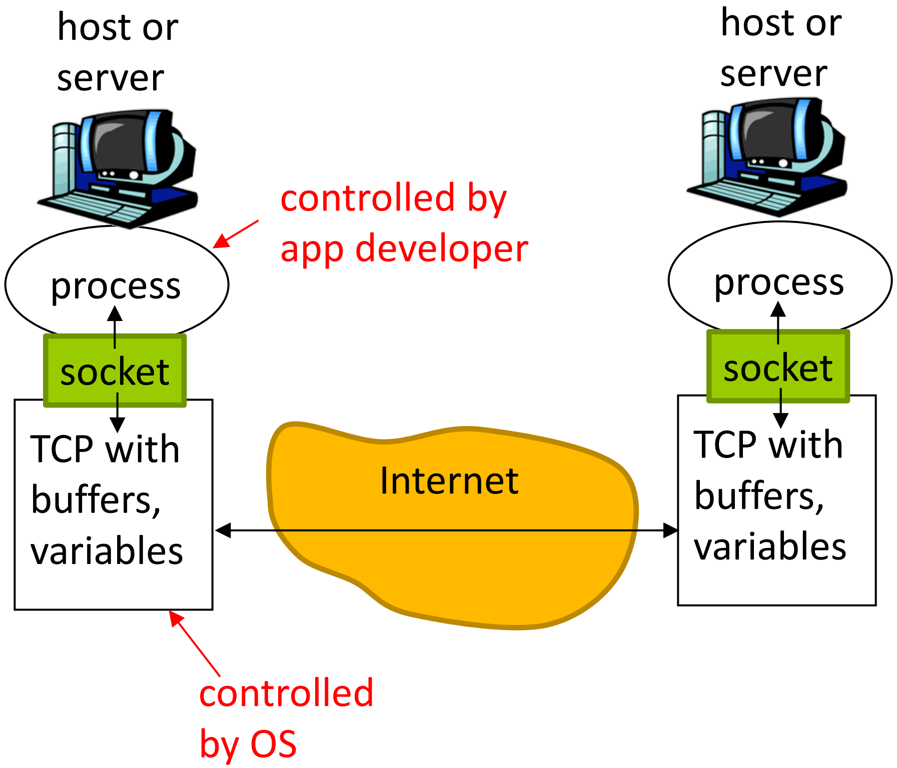
- Overview of networking and layered architecture
- Wireshark packet sniffer and Scapy packet manipulation
- Wired LAN, Wireless LANs, VLANs
- IPv4, IPv6 ARP, ICMP
- UDP
- DHCP

Today's Topics

- HyperText Transport Protocol (HTTP)

What is a Socket?

- Process sends/receives messages to/from its socket
- Socket analogous to door
  - Sending process shoves message out door
  - Transport infrastructure on other side of door carries message to socket at receiving process
  - **Imagine you are just writing to a file…**
- API allow customization of socket
  - Choose transport protocol
  - Choose parameters of protocol


host or server
host or server
controlled by app developer
process
process
socket
socket
TCP with buffers, variables
TCP with buffers, variables
Internet
controlled by OS

Application-Layer Protocol

- Sockets just allow us to send raw messages between processes on different hosts
  - Transport service takes care of moving the data
- **What** exactly is sent is up to the application
  - An application-layer protocol
  - HTTP, IMAP, Skype, etc…

Application-Layer Protocol

- Both the client and server speaking the protocol must agree on
  - Types of messages exchanged
    - e.g., request, response
  - Message syntax
    - What fields are in messages
    - How fields are delineated
  - Message semantics
    - Meaning of information in fields
  - Rules for when and how processes send and respond to messages

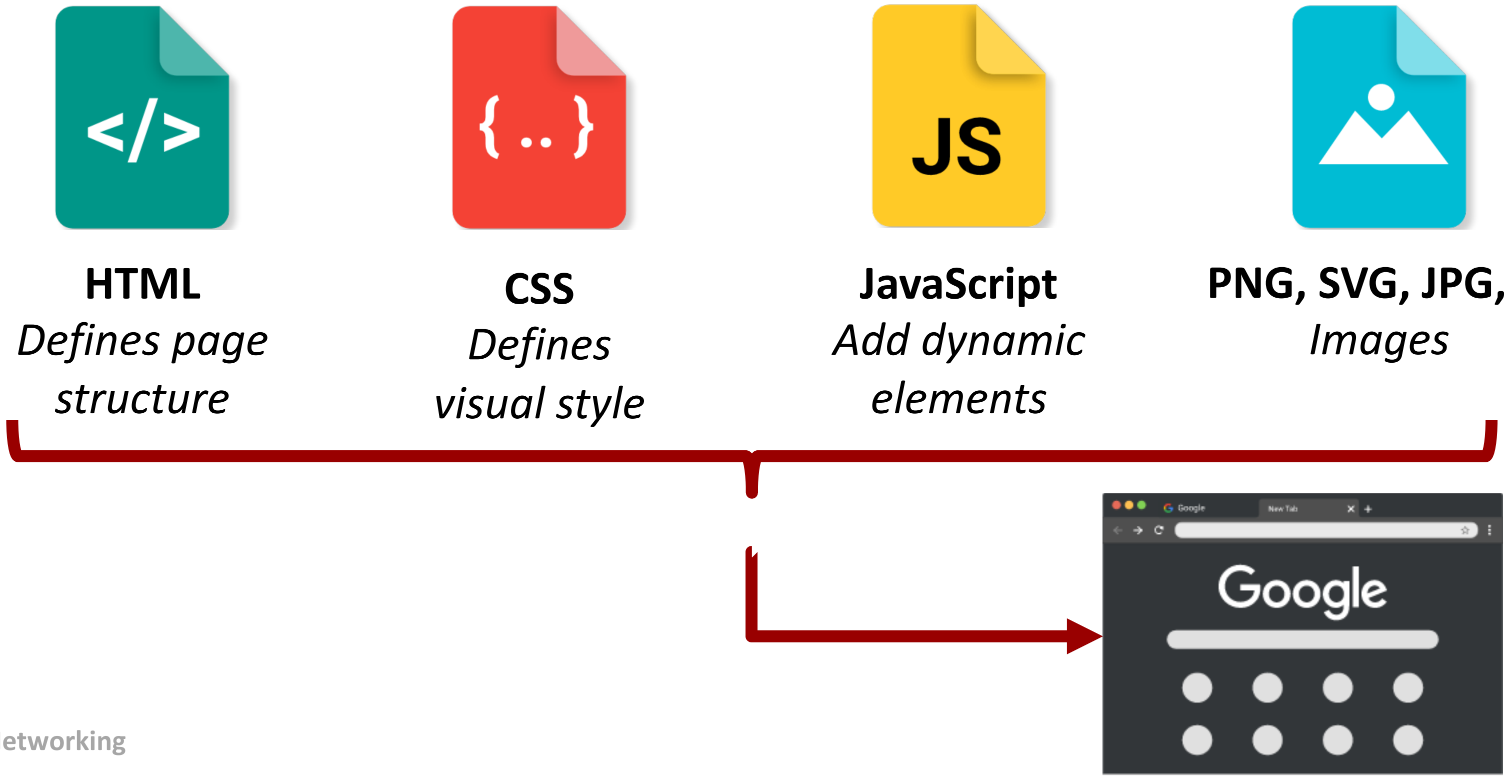
Application-Layer Protocol

- **Public-domain** protocols:
  - Defined in RFCs (Request for Comment)
  - Allows for interoperability
  - Examples: HTTP, SMTP, BitTorrent
- **Proprietary** protocols
  - Examples: Skype

Hypertext Transport Protocol (HTTP)

Web Pages

- Web **page** consists of base HTML file and (potentially) many referenced **objects**



HTML
Defines page structure

CSS
Defines visual style

JavaScript
Add dynamic elements

PNG, SVG, JPG, …
Images

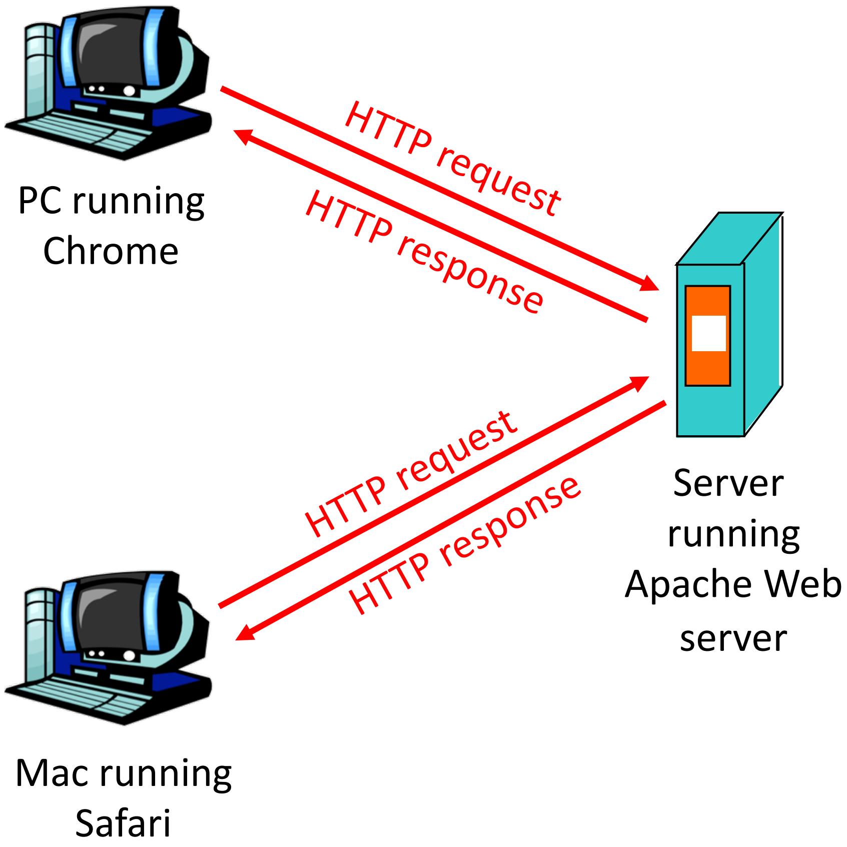
Web URLs

- Each object (HTML, CSS, JS, etc...) is addressable by a URL
- Example:

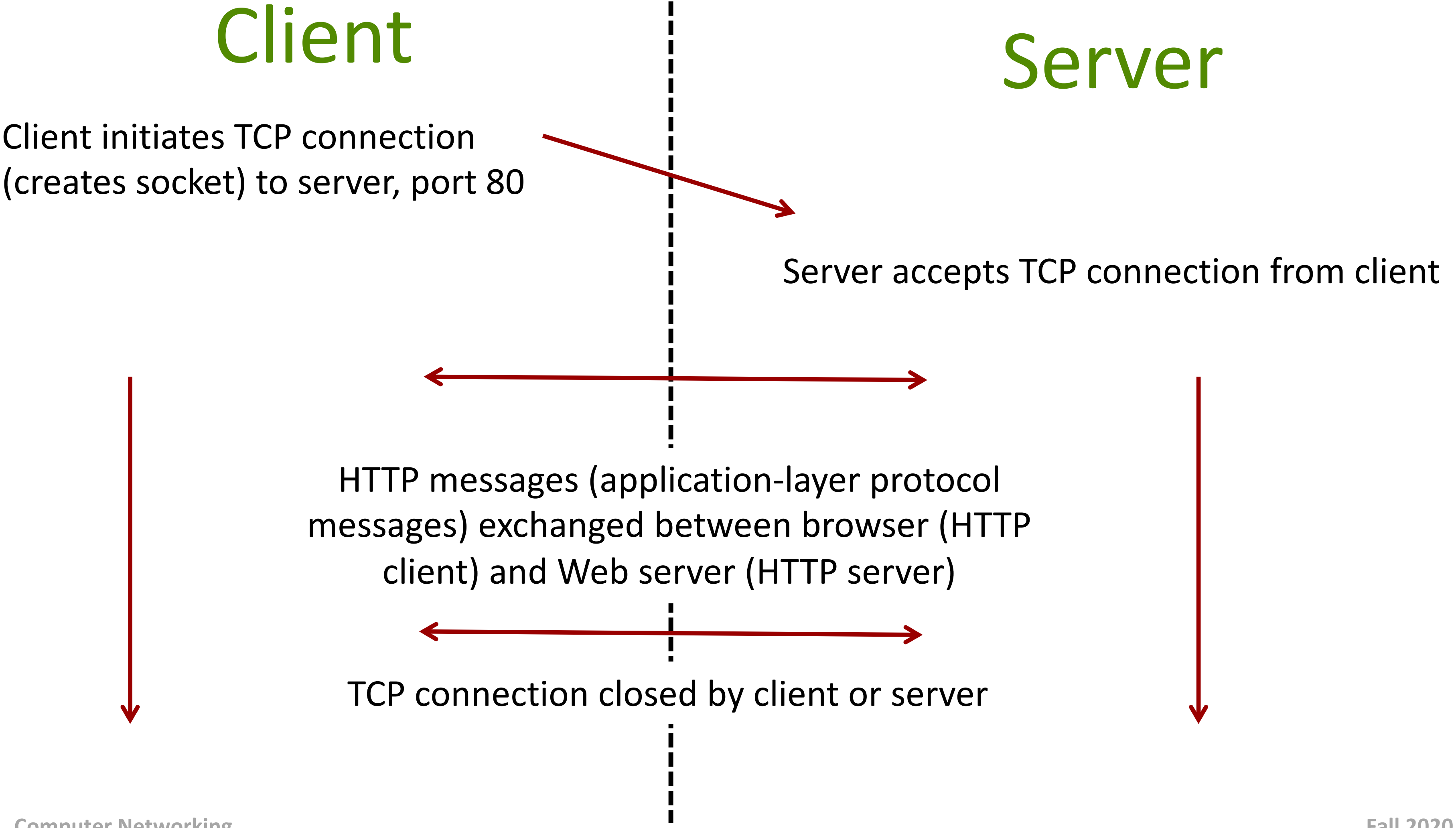
$$\underbrace{\texttt{cyberlab.pacific.edu}}_{\textbf{Host name}}\texttt{/}\underbrace{\texttt{comp177/header.png}}_{\textbf{Path name}}$$

Hypertext Transfer Protocol Overview

- **HTTP** is the *application layer protocol* for the web
- It is how the client and server communicate
- Client/server model
  - **Client**: browser that requests, receives, "displays" Web objects
  - **Server**: Web server sends objects in response to requests



HTTP Overview



Client

Server

Client initiates TCP connection (creates socket) to server, port 80

Server accepts TCP connection from client

HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)

TCP connection closed by client or server

HTTP Overview

- HTTP is “stateless”
- Server maintains no information about past client requests

- Why no state?
  - Protocols that maintain “state” are complex!
  - Past history (state) must be maintained
  - If server/client crashes, their views of “state” may be inconsistent and must be reconciled

HTTP Connections

Non-persistent HTTP

- At most one object is sent over a TCP connection
- Single request, single response

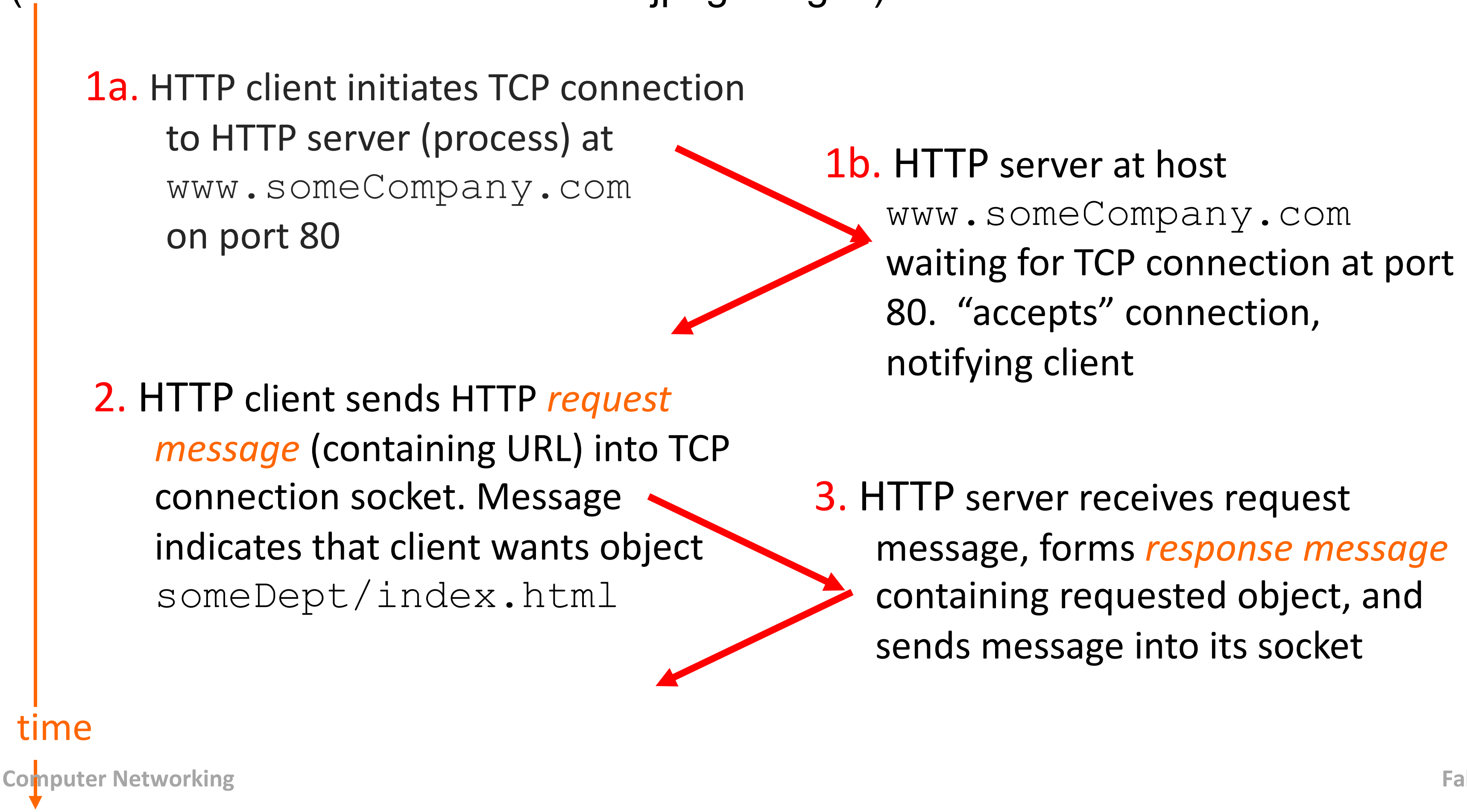
Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server
- Single request, multiple responses

Nonpersistent HTTP

Suppose user enters URL `www.someCompany.com/someDept/index.html`

(contains text and references to 10 jpeg images)



1a. HTTP client initiates TCP connection to HTTP server (process) at `www.someCompany.com` on port 80

1b. HTTP server at host `www.someCompany.com` waiting for TCP connection at port 80. “accepts” connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDept/index.html`

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

Nonpersistent HTTP

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

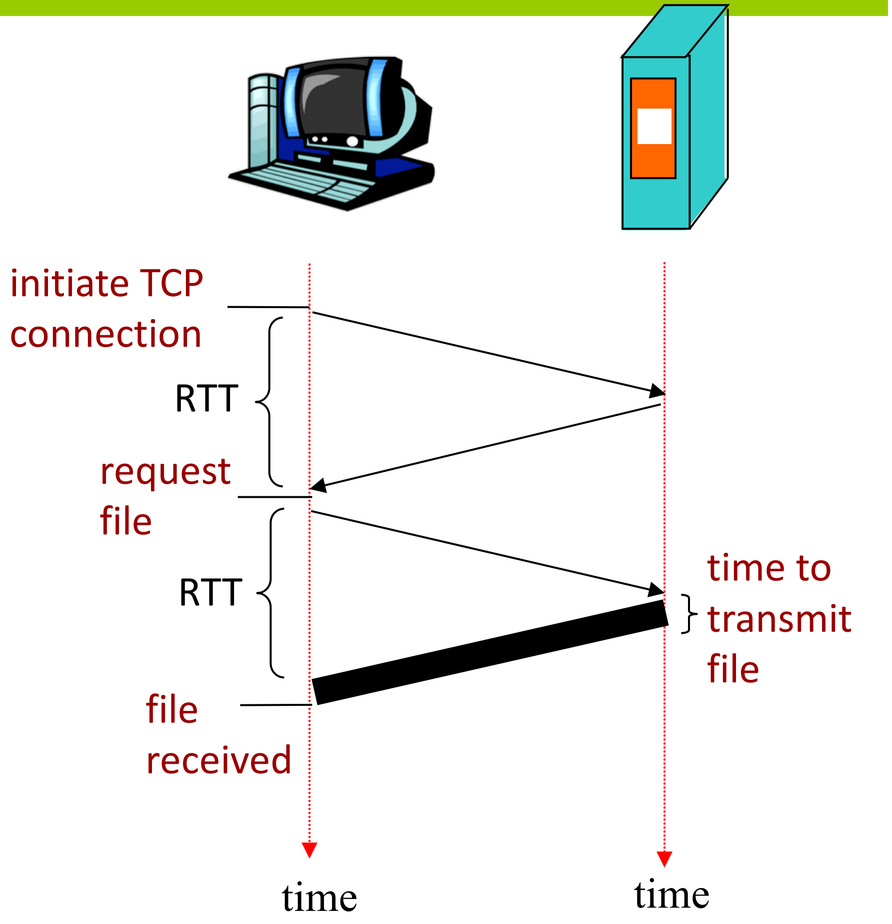
time

6. Steps 1-5 repeated for each of 10 jpeg objects

Why is this approach considered slow?

Non-Persistent HTTP: Response Time

- **RTT (Round Trip Time):**
  - Time for a small packet to travel from client to server and back.
- **Response time:**
  - One RTT to initiate TCP connection
  - One RTT for HTTP request and first few bytes of HTTP response to return
  - File transmission time
- **Total = 2RTT+transmit time (per object!)**



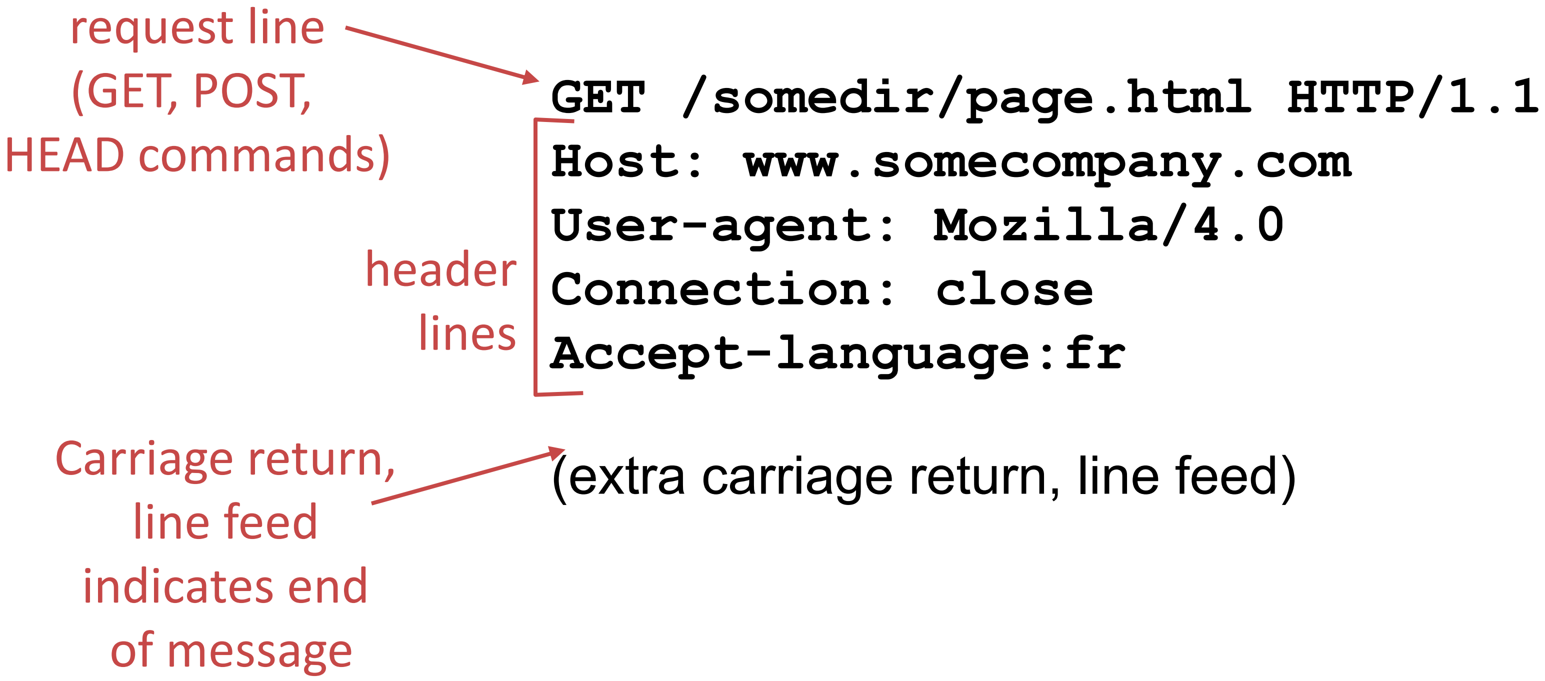
Persistent vs Non-Persistent HTTP

- **Non-Persistent HTTP issues**
  - Requires 2 RTTs per object
  - OS overhead for each TCP connection
  - Browsers often open parallel TCP connections to fetch referenced objects (more overhead)

- **Persistent HTTP**
  - Server leaves connection open after sending response
  - Subsequent HTTP messages between same client/server sent over open connection
  - Client sends requests as soon as it encounters a referenced object
  - As little as one RTT for all the referenced objects

HTTP Request Message

- HTTP request messages
  - Used to send data from client to server
  - ASCII (human-readable format)



request line (GET, POST, HEAD commands)

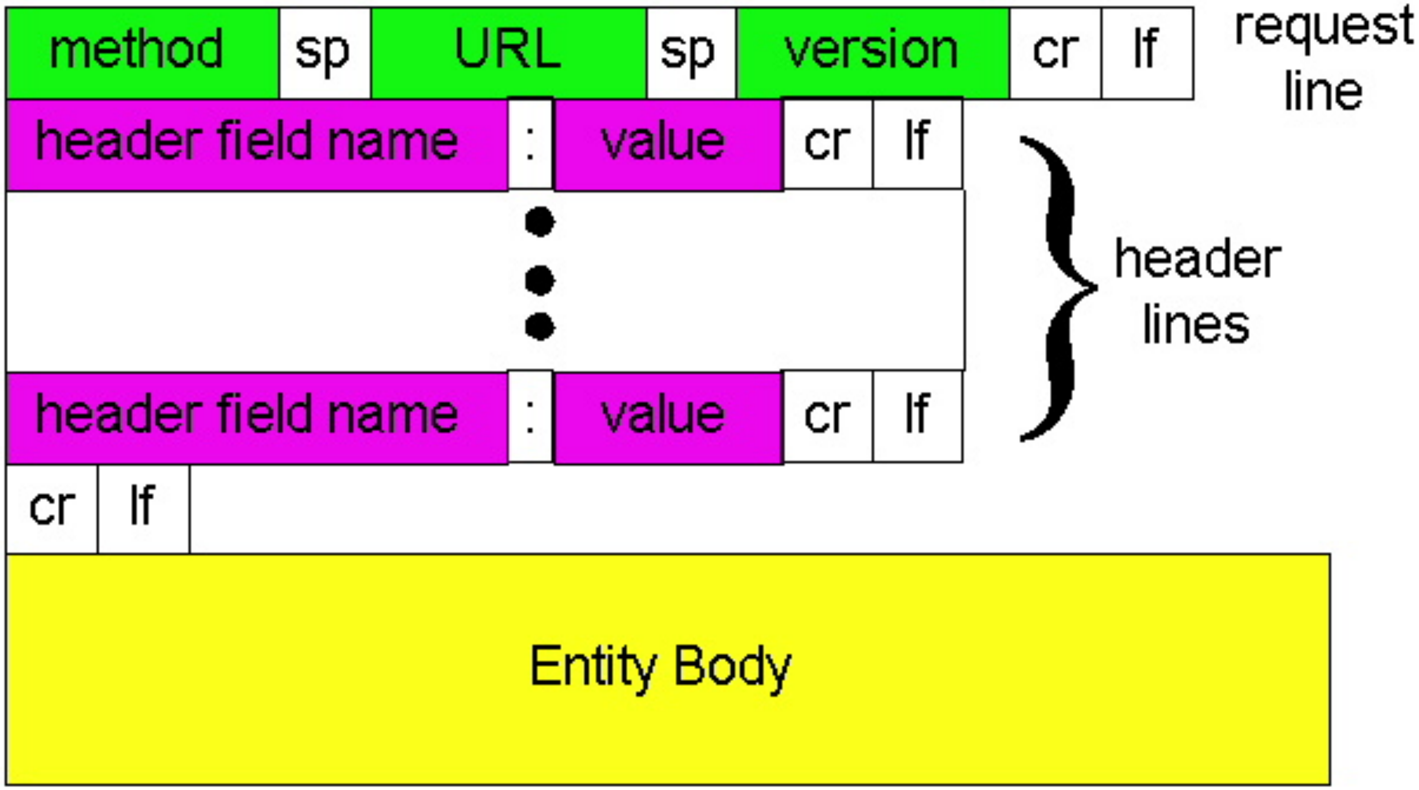
```
GET /somedir/page.html HTTP/1.1
Host: www.somecompany.com
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP Request Message: General Format



Uploading Form Input

- **Post method**
  - Web page often includes form input
  - Input is uploaded to server in entity body
- **URL method**
  - Uses GET method
  - Input is uploaded in URL field of request line

```
www.somecompany.com/page.php?variable1=testData
```

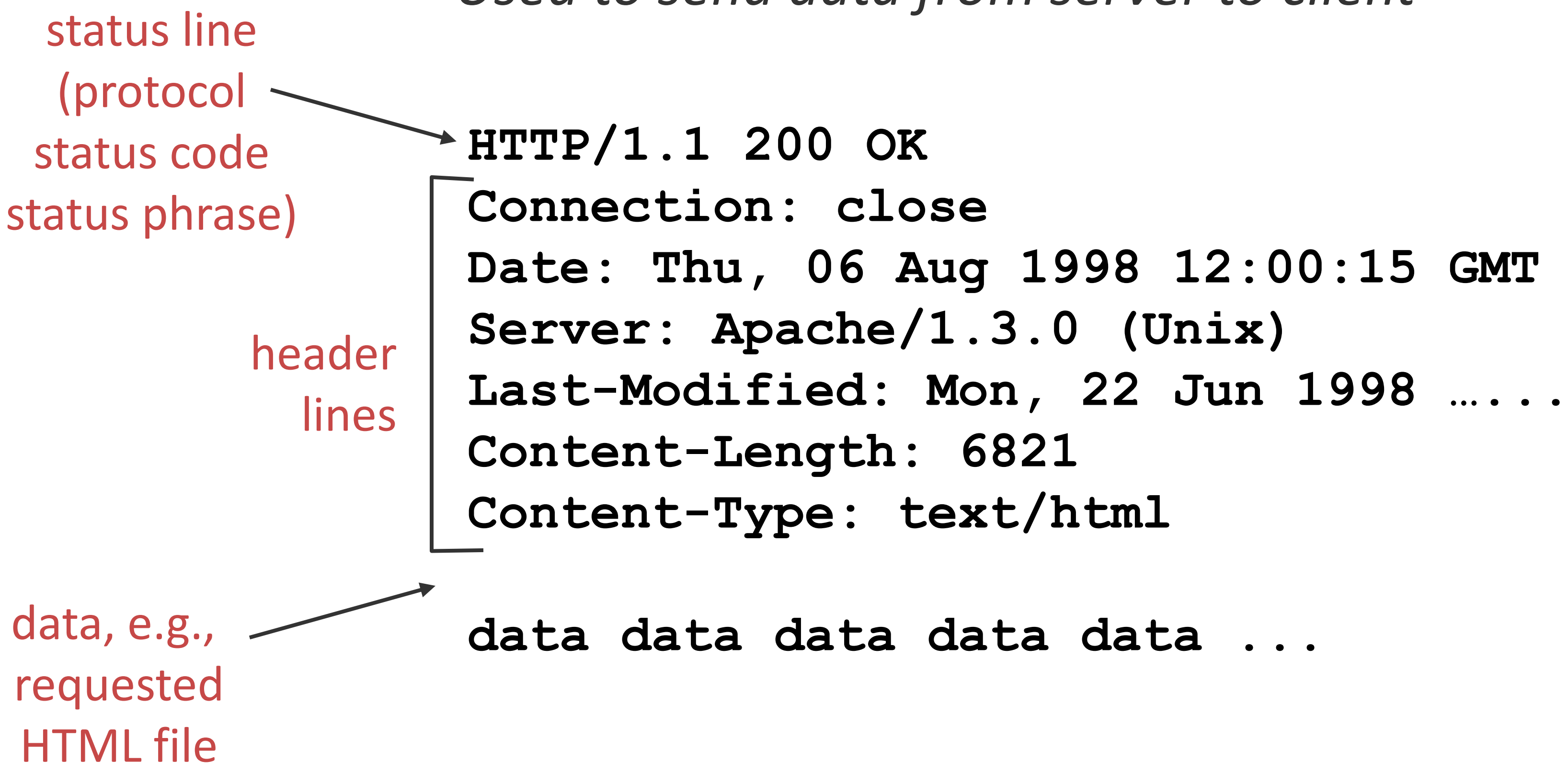
Method Types

- **HTTP/1.0**
  - GET
    - Retrieve object from server
  - POST
    - Upload object to server
  - HEAD
    - Retrieve *only the header* associated with an object (not the object itself)

- **HTTP/1.1**
  - GET, POST, HEAD
  - PUT
    - uploads file in entity body to path specified in URL field
  - DELETE
    - deletes file specified in the URL field

HTTP Response Message

Used to send data from server to client



status line (protocol status code status phrase)

header lines

data, e.g., requested HTML file

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP Headers (Common for Requests)

- A few examples (out of many!)
  - `User-Agent`: Type of web browser (Family? Version? Mobile?)
  - `Host`: Domain name of site (and optional port number)
    - Required for HTTP/1.1
    - Required to support multiple sites hosted on same server
  - `If-Modified-Since`: Used in conditional requests
    - *Only send the file if it has been modified AFTER a certain date. Otherwise, server responds with* `Not Modified`
  - `Referrer`: URL of page client visited previously (that has a link to current URL)

HTTP Headers (Common for Replies)

- A few examples (out of many!)
  - **`Server`**: Version/type of web server
  - **`Date`**: Date & Time HTTP response was generated
  - **`Last-Modified`**: Date & time the attached object was last modified
  - **`Content-Length`**: Length of attached object in bytes
  - **`Content-Type`**: Media type of the attached object
    - text/html, image/png, application/javascript
  - **`Content-Encoding`**: Encoding (compression) format of object
    - gzip

HTTP Headers (Persistent HTTP)

- To enable persistent HTTP
  - **`Connection: Keep-Alive`**
    - Tell server - client wants persistent connection
  - **`Connection: close`**
    - Tell client or server – persistent connection not supported, socket will be closed after object
  - **`Keep-alive: timeout=n, max=m`**
    - Tell client
      - **n** = Number of idle seconds before server closes connection
      - **m** = Maximum number of requests within one persistent connection

HTTP Response Status Codes

In first line in server->client response message. A few sample codes:

`200 OK`

- ➚ request succeeded, requested object later in this message

`301 Moved Permanently`

- ➚ requested object moved, new location specified later in this message (Location:)

`400 Bad Request`

- ➚ request message not understood by server

`404 Not Found`

- ➚ requested document not found on this server

`500 Internal Server Error`

Trying out HTTP (Client side) for Yourself

1. Use netcat (nc) to open a TCP socket to your favorite Web server:

```
nc -vc www.google.com 80
```

Opens TCP connection to port 80 (default HTTP server port) at `www.google.com`
Anything typed in sent
to port 80 at `www.google.com`

2. Type in a GET HTTP request:

```
GET /about/ HTTP/1.1
Host: www.google.com
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

Demo Time!

Netcat Demo

- Manual file request

Wireshark Demo

- Filtering on protocol headers
- Viewing request/response
- HTTP conversation analysis of all captured packets

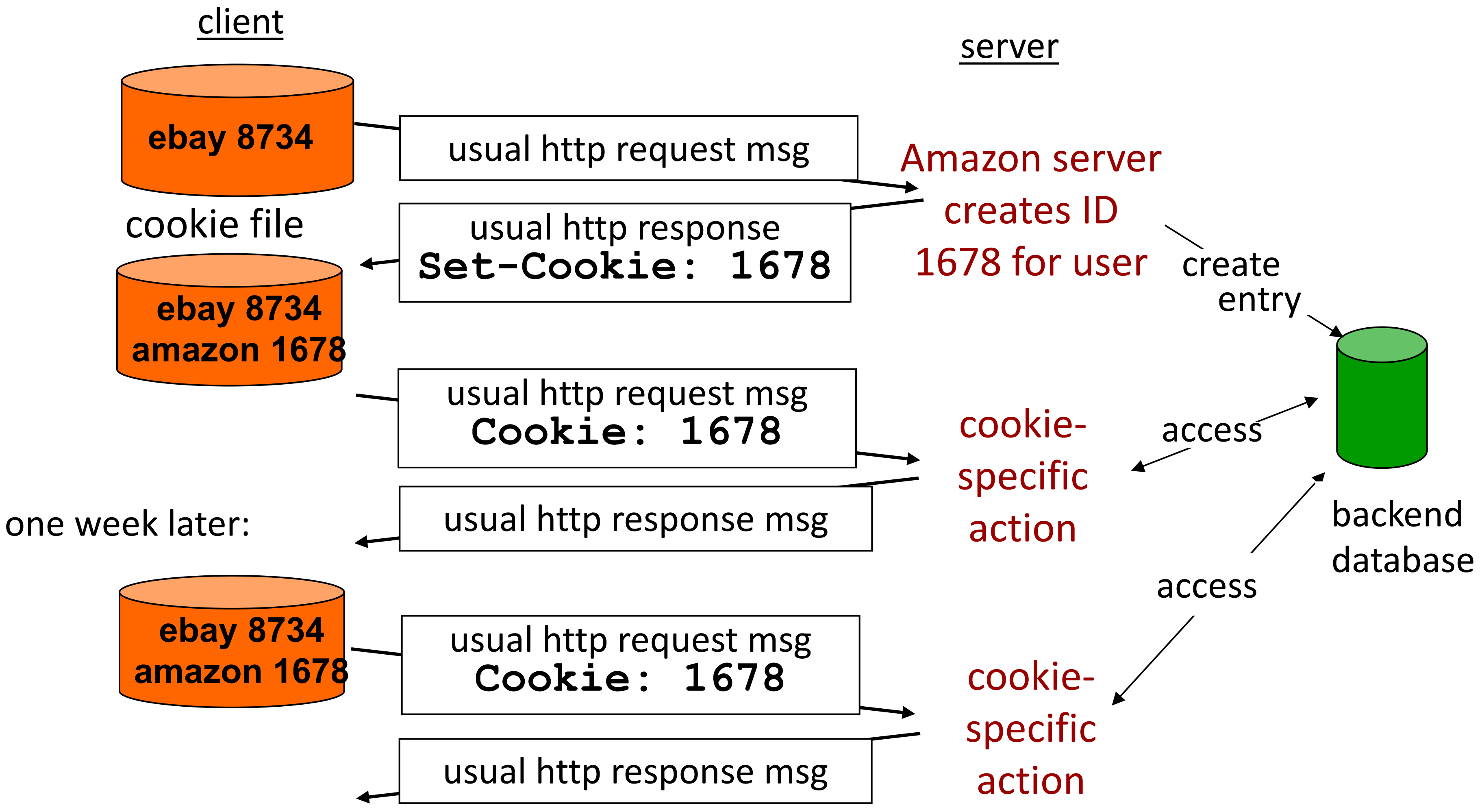
User-Server State: Cookies

- HTTP is stateless
  - State is sometimes desired
- Solution? Cookies!
  - Created when you visit a site for the first time
  - When initial HTTP requests arrives at site, site creates:
    - Unique ID
    - Entry in backend database for ID

- Four components

1. **Cookie header line** of HTTP *response* message
2. Cookie header line in HTTP *request* message
3. Cookie file kept on **user's host**, managed by user's browser
4. **Back-end database** at Web site

Cookies: keeping “state”



client

server

ebay 8734

cookie file

usual http request msg

Amazon server creates ID 1678 for user

usual http response
`Set-Cookie: 1678`

ebay 8734
amazon 1678

create entry

usual http request msg
`Cookie: 1678`

cookie-specific action

access

usual http response msg

one week later:

backend database

access

ebay 8734
amazon 1678

usual http request msg
`Cookie: 1678`

cookie-specific action

usual http response msg

Cookies

- Cookies store **Key -> Value pairs**
- What can I do with this?
  - Authorization, shopping carts, user session state (Web e-mail)
- How to keep “state”:
  - Protocol endpoints (sender/receiver) both have to maintain data over multiple transactions
  - Cookies: http messages carry state
- Tension between users and websites
  - **Websites:** If I can track you, I can make money from marketers
  - **Users:** I don’t want to be tracked (and thus can delete cookies)

Closing Thoughts

Recap

- Today we discussed
  - URLs
  - HTML
  - HTTP

Next Class

- TCP

Class Activity

CA.13 – HTTP & Wireshark

Due tonight at 11:59pm